Static Execution Time Analysis

Niklas Holsti

Space Systems Finland Ltd (now)
Tidorum Ltd (to be)

Overview

• Area of interest
• Current state
• Work in progress
• What to do next
Area of interest

• **Static analysis of programs for**
  – Bounds on execution time and memory space
  – other properties that depend on:
    • the possible execution paths
    • the time/space/energy usage along the execution path
    • the sequence of actions on the execution path (~ protocols)

• **Applications**
  – analysis of executable (binary) programs
  – for embedded real-time systems
  – for verification (meets time and space limits)
  – for understanding (time and space per program part)
Static execution-time analysis

- CPU speed model
- (Sub)program code
- Bounds on input data

Static analysis

- Bounds on exec time

Problem is unsolvable in general <= Halting Problem.
- need restrictions on program structure
- may get pessimistic (safe but inaccurate) results
Current state = the Bound-T tool

- **Analyses worst-case execution time and stack usage**
  - for deterministic processors (no cache, linear pipeline)
    - SPARC V7 (ERC32), ADSP 21020, Intel 8051, ARM7 (proto)
  - from compiled, linked binary (no source-code analysis)

- **Implementation**
  - manually written (Ada 95)
  - modular: target-specific part + generic part

- **Generic techniques**
  - program model = flow-graphs + call-graph + assertions
  - loop counters modelled by Presburger arithmetic (Omega tool)
  - worst-case execution path from ILP (lp_solve tool)
  - assertion language using syntactic structure of program
Bound-T flow

Source code ➔ Compile ➔ Link ➔ Exe

Assertions ➔ Bound-T ➔

- WCETs
- Call Tree
- Stack bounds
- HRT Execution Skeleton
Work in progress

• *Increasing power of arithmetic analysis*
  – Constant propagation to simplify program model
  – slicing along dependencies to simplify program model
  – optimized translation to Presburger formulae

• *Increasing power of flow analysis*
  – Less constrained loop structures (DJ method)

• *Better analysis of dynamic addresses*
  – case/switch statements, jump tables
  – array accesses, pointers to data or code

• *More powerful assertions*
  – context-dependent (call-path dependent) assertions

• *Porting to more target processors*
EU research cooperation

• **ARTIST 2 Network of Excellence**
  – proposal for EU 6th Framework Program
  – cluster: “Compilers and Timing Analysis” led by R. Wilhelm
  – participants: most EU WCET research groups
    • Saarbrücken, AbsInt, Mälardalen, TU Wien, IRISA, York, SSF, ...
  – aims defined by “integration” purpose of NoE:
    • define common modular structure of WCET tools
    • interoperation of modules from various sources
    • adapt existing academic & commercial tools to conform
  – preparation for a larger FP6 WCET proposal in mid-2004

• **ForTIA = Formal Techniques Industry Association**
  – Mainly specification & verification tools, little analysis
What to do next in R & D

• **Feasible paths**
  – theory? representation? analysis? presentation? ...

• **Loops**
  – nested loop dependencies, eg. triangular loops
  – inter-loop dependencies
  – non-counting loops: shifting loops, binary search, ...

• **Dynamic processor architectures**
  – caches, parallel units, multiple issue, ...

• **Generative implementation of target-specific analysis modules**
  – languages to describe target processors
  – trade-off: language power $\iff$ implementation complexity
Example of feasible path problem (real case!)

procedure A is
begin
  for n in 1 .. 200 loop
    B (action(n), ok);
    exit when ok;
  end loop;
end A;

procedure B
  (act : in action_t; ok : out boolean) is
begin
  Quick_Try (act, ok);
  if ok then
    Long_Comp (act);
  end if;
end B;

• Expected WCET(A, B) ~ 20 ms
• Syntactic paths (A, B) => Long_Comp 200 times => 4 seconds!
• Feasible paths (A, B) => Long_Comp once => 20 ms.
This one could be solved by different design

procedure A is
begin
    for n in 1 .. 200 loop
        Quick_Try (action(n), ok);
        if ok then
            Long_Comp (action(n));
            exit;
        end if;
    end loop;
end A;

• Syntactic paths \((A, B)\) = Feasible paths \((A, B)\)
  \(\Rightarrow\) Long_Comp once \(\Rightarrow\) 20 ms.
• Perhaps “inlining” during analysis would see this, too.
Research problems in feasible paths analysis

• **Formal representation**
  – ? similar to flow graphs, or very different (other “aspects”)
  – ? enumerative, linguistic, algebraic, automata, ...

• **Analysis**
  – ? how: discover variable relationships, condition dependencies, ...
  – ? what: find the important path constraints, ignore trivial ones

• **Generality and usefulness**
  – ? same or different path representation & analysis for
    • time analysis
    • memory analysis
    • points-to analysis
    • functional correctness & proof
    • etc.
The End

or the beginning?